(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 712 076 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
15.05.1996 Bulletin 1996/20

(51) Int Cl.[6]: **G06F 9/46**, G06F 15/16

(21) Application number: **96200086.5**

(22) Date of filing: **22.11.1991**

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IT LI LU NL SE

(30) Priority: **14.02.1991 US 655296**

(62) Application number of earlier application in accordance with Art. 76 EPC: **92901473.6**

(71) Applicant: **CRAY RESEARCH, INC.**
Eagan, Minnesota 55121 (US)

(72) Inventor: **Schiffleger, Alan J.**
Chippewa Falls, Wisconsin 54729 (US)

(74) Representative:
Beresford, Keith Denis Lewis et al
BERESFORD & Co.
2-5 Warwick Court
High Holborn
London WC1R 5DJ (GB)

Remarks:
This application was filed on 15 - 01 - 1996 as a divisional application to the application mentioned under INID code 62.

(54) System for distributed multiprocessor communication

(57) A tightly coupled communication scheme based on a common shared resource circuit and adapted particularly to a multiprocessing system including 2^N CPUs. A mechanism has been added that allows data in a shared register to be read and incremented as a single instruction, eliminating the need for semaphore manipulations during the operation. A second mechanism has been added to permit the use of indirect addressing in the addressing of semaphore bits and shared registers. Operating systems can relocate semaphore bits and message areas to permit simultaneous execution of the same function within a single task. In addition, an instruction has been added which tests of the semaphore bit and acts upon the state of that bit. If the semaphore bit is not set then the processor takes control of the semaphore bit by setting it. If the semaphore bit is set, the processor will execute a branch and execute other instructions. Thus, jobs assigned to a processor in a multiprocessing, multitasking application do not block or wait for the semaphore bit to clear.

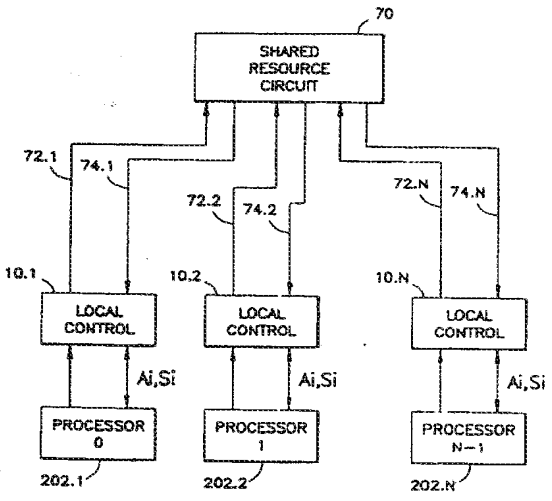


FIG. 1

EP 0 712 076 A2

8NSDOCID: <EP 0712076A2 I >

Description

Background of the Invention

Field of the Invention

The present invention pertains to the field of high speed digital data processors and more particularly, to communication between processors in a multiprocessor system.

Background Information

Interprocessor communication is an important factor in the design of effective multiprocessor data processing systems for multitasking applications. System processors must be able to execute independent tasks of different jobs as well as related tasks of a single job. To facilitate this, processors of a multiprocessor system must be interconnected in some fashion so as to permit programs to exchange data and synchronize activities.

Synchronization and data transfers between independently executing processors typically are coordinated through the use of controlled access message boxes. A single bit semaphore is used to prevent simultaneous access to the same message box. In operation, a processor tests the state of the semaphore bit. If the semaphore bit is set, the message box is currently "owned" by another processor. The requesting processor must then wait until the semaphore is cleared, at which time it sets the semaphore and can access the message box.

A typical approach to interprocessor communication in prior art machines was to use main memory as the location of the message boxes and their associated semaphore bits. This "loosely coupled" approach minimizes interprocessor communication links at the cost of increasing the overhead for communications. However when the number of processors in a multiprocessing system increases, processors begin to contend for limited resources. For instance, accessing a "global" loop count stored in main memory and used to track iterations of a process executed by a number of different processors is relatively simple when there are only two or three processors. But in a loosely coupled system a processor's access to a global loop count contends with other processors' accesses to data in memory. These contentions delay all memory requests.

A different approach was disclosed in U.S. Patent No. 4,636,942 issued to Chen et al. and in U.S. Patent No. 4,754,398 issued to Pribnow, both of which patents are hereby incorporated herein by reference. The above documents disclose "tightly coupled" communication schemes using dedicated "shared" registers for storing data to be transferred and dedicated semaphores for protection of that data. Shared registers are organized to provide N + 1 "clusters" where N equals the number of processors in the system. Clusters are used to restrict access to sets of shared registers. Processors are assigned to a cluster as part of task initialization and can access only those shared registers that reside in their cluster. A semaphore register in each cluster synchronizes access to cluster registers by processors assigned to the same cluster.

Tightly coupled communication schemes reduce communication overhead by separating interprocessor communication from the accesses to memory that occur as part of the processing of a task. However, even in tightly coupled systems, communication overhead increases as a function of the number of processors in a system. This increased overhead directly impacts system performance in multitasking applications. A large number of processors contending for a piece of data (such as a global loop count) can tie up even a dedicated cdmmunications path due to increased message traffic. This has been recognized and steps have been proposed to streamline communications in a tightly coupled system.

Patent No. 4,754,398 discloses a method for reducing interprocessor communication traffic incurred in executing semaphore operations in a tightly coupled system. A copy of a cluster's global semaphore register is kept in a local semaphore register placed in close proximity to each processor in the cluster. Operations on a cluster's global semaphore register are mirrored in operations on the local semaphore registers associated with that cluster. The use of a local semaphore register reduces the delay between the issuance of a semaphore test command and the determination of the state of that semaphore.

Commonly owned, copending application No. 07/308,401 by the present inventor goes a step further by streamlining the local semaphore testing and by replacing the shared real time clock circuit with distributed local real time circuits. That application also extends the tightly coupled design to a system of eight processors. It is hereby incorporated by reference.

In the above system the shared semaphore and information register circuit is partitioned such that one byte of the 64 bit interprocessor communication system is located on each processor board. The bytes are distributed such that the least significant byte of each information register resides on CPU0 and the most significant byte on CPU7. Interprocessor communication commands are a single byte in length; these commands are replicated at the source so as to send the same command byte to each shared circuit in the system.

Global semaphore registers for the above system are distributed among the processors. Since each semaphore register is only 32 bits wide, the least significant byte of each semaphore register is kept on CPU4 and the most significant byte is kept on CPU7.

A local control circuit is placed on each processor board. This circuit receives a interprocessor communication instruction from the processor on the board and determines when to issue the instruction to the shared

communication circuitry. In addition, the control circuit knows the cluster that the processor is assigned to and keeps a copy of the semaphore register associated with that cluster in its local semaphore register.

By software convention, a CPU wishing to access a shared information register must gain control of the semaphore associated with that register. First, the CPU issues a Test_and_Set instruction on the semaphore. If the bit is set, the local circuit halts the CPU until the bit clears and there are no other higher priority interprocessor communication requests. The local circuit then allows issue of the Test_and_Set instruction and the proper semaphore is set in the shared semaphore register and in each local semaphore register assigned to that cluster.

Once the semaphore bit is set the CPU can access its associated information register by issuing a Shared_Register_Read or Shared_Register_Write instruction. Upon completion of the necessary operations on the shared register, the CPU clears the semaphore bit in the shared semaphore register and the proper bit in the local semaphore registers assigned to that cluster are cleared. While the semaphore bit is set no other processor can access the associated information register.

As the number of processors increase, the methods disclosed to date are not adequate to meet the needs of systems having an increased number of processors. The steps required to access and control global variables such as loop counts stored in shared registers adds a significant burden to communications overhead. In the meantime, access to these registers by other processors in the cluster is not permitted. Processors requiring access to the loop count must wait until the semaphore bit is cleared. This has the potential to waste a considerable amount of CPU time.

It is clear that further changes are necessary in the design of a tightly coupled communication circuit to achieve reduced message traffic.

Summary of the Invention

The present invention is an implementation of a tightly coupled communication scheme adapted particularly to, but without limitation thereto, a system including 16 CPUs. According to the present invention a mechanism has been added that allows data in a shared register to be read and incremented as a single instruction, thus eliminating the need for semaphore manipulations during the operation.

According to another aspect of the present invention, an instruction has been added which tests of the semaphore bit and acts upon the state of that bit. If the semaphore bit is not set then the processor takes control of the semaphore bit by setting it. If the semaphore bit is set, the processor will execute a branch and execute other instructions. Thus, the job does not block or wait for the semaphore bit to clear.

According to yet another aspect of the present invention, a mechanism has been added to permit the use of indirect addressing in the addressing of semaphore bits and shared registers. Operating systems can relocate semaphore bits and message areas to permit simultaneous execution of the same function within a single task.

Brief Description of the Drawings

Fig. 1 is a high-level block diagram of a tightly coupled multiprocessor system according to the present invention.

Fig. 2 is a block diagram of the common shared register resource circuitry according to the present invention.

Fig. 3 is a simplified schematic block diagram of the local shared register access circuitry according to the present invention.

Fig. 4 is a table illustrative of a write operation according to the present invention.

Fig. 5 is a table illustrative of an I/O channel operation according to the present invention.

Detailed Description of the Preferred Embodiment

In the following detailed description of the preferred embodiment, references made to the accompanying drawings which form a part thereof, and which is shown by way of illustration a specific embodiment in which the invention may be practiced. The preferred embodiment of the present invention is designed to operate within a tightly coupled multiprocessor system of sixteen processors. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

FIG. 1 illustrates a high-level block diagram of the tightly coupled multiprocessor communication system 200 within a multiprocessor data processing system. Processors 202.1 through 202.N are connected to local control circuits 10.1 through 10.N, respectively. Local control circuits 10.1 through 10.N are connected in turn through shared register write paths 72.1 through 72.N and shared register read paths 74.1 through 74.N to shared resource circuit 70. In the preferred embodiment, paths 72 and 74 are 64 bits wide. Also, in the preferred embodiment, each local control circuit is placed on its associated processor's circuit board to ensure close proximity. This permits use of a separate instruction path and separate 64 bit address register and scalar register read and write paths to connect processors 202 to local control circuits 10.

Further, in the preferred embodiment shared resource circuit 70 is partitioned by bit-slicing the registers in circuit 70 into N equal subcircuits and duplicating the control circuits so as to create N autonomous subcircuits 71.1 through 71.N. One subcircuit 71 is then placed on

a circuit board with a processor 202 and a local control circuit 10, reducing the number of circuit boards in the system.

FIG. 2 illustrates a shared resource subcircuit 71 for a multiprocessing system containing sixteen processors. Four bit lines from shared register write paths 72.1 through 72.16 are connected through write selector 76 to write registers latch 78. The four bits sent to each subcircuit 71 depend on the processor board that the subcircuit is placed on. In the preferred embodiment, subcircuit 71 placed on Processor M receives bits (M*4) through (M*4)+3 as shown in FIG. 4.

In a like manner, read registers latch 88 is connected to four bit lines from shared register read paths 74.1 through 74.16 for transferring data from the shared resource circuit 70 to the requesting processor 202.

Write registers latch 78 is connected to global shared registers 90, through command demultiplexer 84 to command decoder 80, through read registers selector 92 to read latch 88 and through I/O channel demultiplexer 86 to one or more I/O channels (not shown). I/O channel multiplexer 96 and shared registers 90 are also connected through read registers selector 92 to read latch 88. In addition shared registers 90 are connected to read and increment circuit 94 for automatically incrementing the contents of a register within shared registers 90.

In the preferred embodiment, shared registers 90 are segmented into N+1 clusters of sixteen information registers (eight shared B and eight shared T) and one semaphore register. Shared B registers are used to transfer addresses; shared T registers are used to transfer scalar data. Access to registers within each cluster is limited to those processors 202 that are assigned to that cluster.

Command decoder 80 is connected to write selector 76, shared registers 90, read selector 92 and read and increment circuit 94. Command decoder 80 decodes commands received from local control circuits 10.1 through 10.16 and controls the movement of data within resource subcircuit 71. Command decoder 80 also provides feedback to local control circuits 10.1 through 10.16 so they can modify their local semaphore registers to reflect changes in shared semaphore registers. In addition, command decoder 80 controls operation of the attached I/O channel.

Shared register write paths 72.1 through 72.N transmit commands and data to shared resource register 70. In the preferred embodiment, commands are either eight or twelve bits in length. Therefore, since each subcircuit 71 runs independently, the local control circuit 10 sending the command must replicate and send it to each of the subcircuits 71.1 through 71.N. For the sixteen processor case, the first four bits of a command from processor 202.1 are transferred on write path 72.1 to each subcircuit 71.1 through 71.16 at the same time. Then the next four bits are transferred, followed by the next four bits of command and data if required. Each subcircuit then reconstructs the command using command demultiplexer 84 before presenting the command to command decoder 80.

Local control circuits 10.1 through 10.N arbitrate among themselves to prevent more than one access to shared resource circuit 70 at a time. A local control circuit 10 uses a CPU_In_Progress line 32 to indicate that it has control of shared resource 70. In the preferred embodiment, each shared resource subcircuit 71.1 through 71.N is connected to a CPU_In_Progress line 32 from each local control circuit 10.1 through 10.N. The resulting N*N lines are used by the command decoder 80 on each subcircuit 71 to select (through write selector 76) the write path 72 associated with the requesting processor 202.

FIG. 3 shows an electrical block diagram of the local control circuit 10 of FIG. 1. Issue control 16 is connected to current instruction parcel (CIP) register 12, local semaphore register 18, semaphore selector 22, command generator 20 and, externally, to each of the other control circuits 10 and to shared resource circuit 70. Issue control 16 manages the issuance of instructions having to do with shared resource circuit 70. Through CIP register 12, issue control 16 receives instructions from its respective processor 202. Issue control 16, in turn, acts through semaphore index selector 24 to steer semaphore selector 22 with the contents of either CIP register 12 or of a processor 202 address register. The selected semaphore bit can then be tested by issue control 16 in the execution of a test and set instruction.

Issue control 16 generates a shared resource request 30 to each of the other local control circuits 10 and arbitrates received resource requests 34 from the other local circuits 10. Once it has gained control of shared resource circuit 70, issue control 16 asserts a CPU_In_Progress line 32 to shared resource 70 and causes command generator 20 to generate a command based on the contents of CIP register 12. In the preferred embodiment, the resulting command is multiplexed by command multiplexer 26 into two to three nibbles (four bits each) and sent to each subcircuit 71 of shared resource circuit 70.

Command generator 20 is connected to CIP register 12, to processor 202 address registers and, through command multiplexer 26, to write data selector 44. Write data selector 44 routes data from processor 202 scalar and address registers, from address register multiplexer 47 and from command multiplexer 26 through local write data latch 45 to write data path 72.

Data coming from read path 74 is latched in local read data latch 46. Real time clock 58 is connected to read data latch 46 to facilitate broadcast loading of an arbitrary start time. Read data selector 60 is connected to read data latch 46 directly and through read data demultiplexer 50 and to real time clock 58. Data from read data selector 60 can be stored to local semaphore register 18 or to processor 202 scalar and address registers. Semaphore register 18 can be loaded directly from selector 60 or modified one bit at a time through local

semaphore modifier 14. Local semaphore modifier 14 is connected in turn to command decoder 80 for monitoring activity in the shared semaphore registers.

Issue control 16 controls movement of data through control circuit 10. Instructions are stored in CIP register 12 until issue control 16 determines that shared resource circuit 70 is ready to accept a command. Issue control 16 also controls data output by semaphore index selector 22, write data selector 44 and read data selector 60 through selector control 33.

As in the previously referred to copending application by the present inventor, each processor 202 is assigned a cluster number as part of loading a executable task into the processor. When the task is loaded, processor 202 registers the cluster number and requests and loads the semaphore register associated with that cluster into its local semaphore register 18. From that point on, the local control circuit 10 associated with that processor 202 maintains a copy of the assigned cluster's shared semaphore register in its local semaphore register 18.

Shared semaphore registers are used to synchronize activity and to restrict access to shared information registers. In one typical operation, an access to shared information registers begins with processor 202 issuing a "test and set" command to local control circuit 10. Local control circuit 10 then checks the status of the appropriate bit in its local semaphore register 18. If the bit is set, then another processor has control of that shared register and processor 202 waits for the bit to be cleared. If the bit is not set, local control circuit 10 asserts its CPU_In_Progress line 32 to each of the shared resource subcircuits 71 and sends a command to set the bit in the semaphore register for that cluster.

By software convention, setting a bit in the shared semaphore register grants control circuit 10 access to the associated shared information register. Control circuit 10 then has exclusive control to read or write that register. Upon finishing, control circuit 10 clears the set semaphore bit and another processor can access the register.

In the present invention, a new command has been added to further improve the efficiency of the computing system. Where in past machines a processor such as processor 202 tested a semaphore bit and then was required to wait until it cleared, the new command tests the semaphore bit, returns the status and branches to alternate instructions on determining that the bit is set. This frees up CPU cycles that were otherwise wasted waiting for access to a shared register shared by many CPUs.

This new "test and set or branch" instruction is useful at the operating system level in providing alternatives to just sitting and waiting for a system resource to free up. In previous systems, if two CPUs attempted to use the system resource, one CPU would gain control of the resource and the other would wait until it was finished. With the new instruction the second CPU can test for availability of the system resource. If the resource is busy, it can continue performing operating system functions. This permits a polling approach to system resources rather than the previous "get it or wait" approach.

Semaphore registers are 32 bits wide. To test a bit in local semaphore register 18, the contents of CIP register 12 are used to steer the appropriate bit through semaphore bit selector 22 to issue control 16. If the bit is clear, issue control 16 asserts a shared resource request 30 to each local control circuit 10 and compares its request to requests 34 received from other local control circuits 10. In the preferred embodiment, it has been determined that optimal access to shared resource circuit 70 is obtained when priority in accessing shared resource circuit 70 is granted to the processor 202 with the lowest CPU number while requiring that a processor 202 cannot assert a request as long as there is an active request 34 pending from a processor 202 with a higher CPU number. That is, in a sixteen processor system, CPU15 has the highest priority in making a request while CPU0 has the highest priority in getting an active request served. This provides an equal opportunity for all processors 202 to access shared resource 70. Once a request line 30 is set it remains set until the circuit 10 has completed its function, for example, until the data is transferred in a write operation or until the control information including the register address has been transferred to circuit 70 in a read operation.

Once a processor 202 has obtained access to the shared registers, command generator 20 is activated by issue control 16 to generate, in accordance with the operation specified in CIP register 12, two to three nibbles of command. This command is sent to each resource subcircuit 71 where it is received by command decoder 80 and used to control and accomplish the sought after operation. Command multiplexer 26 takes the first nibble generated by command generator 20 and sends sixteen replicas of that nibble on the sixty four bit wide write path 72. This is followed in subsequent clock periods by sixteen replicas of the remaining command nibbles. The active CPU_In_Progress line 32 causes command decoder 80 on each subcircuit 71 to select the write path 72 associated with the processor 202 controlling the shared register access. Each write registers latch 78 of each of the subcircuits 71 of FIG. 2 simultaneously receives the first four bits of the command followed in subsequent clock periods by the remaining nibbles. The command nibbles are reconstructed into a command in command demultiplexer 84 and presented to command decoder 80 for disposition. The command decoder 80 on each subcircuit 71 thus each simultaneously receives the control information necessary to control shared register access and, in particular, the addressing of the shared registers in shared registers 90.

In the preferred embodiment of the present invention, shared register and real time clock commands are two nibbles each. I/O, semaphore and cluster number

commands are three nibbles each.

An example of a read operation will be described. As mentioned above, access to a shared register typically begins with a "test and set" instruction aimed at gaining control of the register. The local control circuit 10 associated with that processor 202 receives the instruction. It checks the local semaphore bit. If the bit is clear, control circuit 10 checks to see if a processor with higher CPU number has a request pending. If so, issue control 16 waits until the request clears before generating its own request. If not, issue control 16 generates a request. Next, issue control 16 checks its request against requests pending by other processors with a lower CPU number. If there are requests from processors with lower CPU numbers pending, issue control 16 waits until those requests clear. Once there are no requests from processors with lower CPU numbers, issue control 16 sets the CPU_In_Progress line 32 to each of the subcircuits 71 and activates command generator 20 to generate a command based on the contents of CIP register 12. The command generated contains the location of the bit in the semaphore register that is to be set. Multiplexer 26 replicates the three nibbles of the command and broadcasts them to each subcircuit 71 in successive clock periods.

Each subcircuit 71 contains a list of the clusters and the processors currently assigned to each cluster. This list is updated each time a processor is assigned to a new cluster. The command decoder 80 in each subcircuit 71 decodes the command and sets the appropriate bit in the shared semaphore register associated with the cluster the processor is assigned to. In addition, each command decoder 80 generates a signal to each local semaphore modifier 14 assigned to that cluster so that the copy of the shared semaphore register in its local semaphore register 18 is updated.

Once the semaphore bit is set, processor 202 issues a "read registers" instruction. The local control 10 generates a request as above. Once it has gotten control of shared resource 70, issue control 16 sets the CPU_In_Progress line 32 to each of the subcircuits 71 and activates command generator 20 to generate a command based on the contents of CIP register 12. The two nibble command includes the address of the desired register in shared registers 90. Multiplexer 26 again generates two nibbles that are sent to each subcircuit 71 in successive clock periods. Command decoder 80 in each subcircuit 71 decodes the command, reads the addressed register in the cluster the processor is assigned to, and writes the contents to read latch 88. Read latch 88 on each subcircuit 71 writes its four bit nibble to read path 74.1 through 74.N such that the four bits from each subcircuit 71 combine to form a single sixty-four bit word on each read path 74. This word is latched into read data latch 46 on the requesting local control circuit 10 and sent through selector 60 to the appropriate scalar or address register.

In a like manner, a write operation is performed on shared registers 90 beginning with distribution of the two control nibbles to each subcircuit 71 but followed on the next succeeding clock period by transmission of data from a selected address register A_i, a selected scalar register S_i or the output of multiplexer 47. A write operation for a sixteen processor system is illustrated in FIG. 4. Since four bits of write path 72 are connected to each subcircuit 71, four bits of the sixty-four bit data word are written into write latch 78 and from there into shared registers 90. As can be seen in FIG. 4, in the first clock period, the four least significant bits of the command are transferred to the subcircuit 71 located on each processor board. In the next clock period, the remaining four bits of the command are transferred and in the following clock period the word to be written is transferred, with the bits distributed as shown in FIG. 4. Again, the destination cluster is determined by looking at the list of processor cluster assignments and the destination register is determined from the command.

The present embodiment permits indirect addressing of registers in shared resource 70. The ability to reassign registers is useful because operating systems can relocate semaphore bits and message areas to permit simultaneous execution of the same function within a single task.

In the preferred embodiment, instructions issued by processor 202 for shared resource access contain a three bit j field and a three bit k field. In previous machines the k field was concatenated to the end of the two least significant bits of the j field to form a pointer to the location of the semaphore bit for a semaphore instruction. This convention is still used in the present embodiment on semaphore instructions in which the most significant bit j_2 is cleared. However, if the most significant bit j_2 of the j field is set indirect addressing is enabled. This means the k field becomes a pointer to an address register A_k. Address register A_k then contains the location of the semaphore bit that is to be acted upon.

In a like manner, in previous machines the j field was used to form an address to a register in the shared resource circuit for a register instruction. If the least significant bit k_0 of the k field is cleared in an instruction according to the present embodiment, this convention still holds. However, if the least significant bit k_0 of the k field is set in a register instruction, the j field forms a pointer to an address register A_j. Address register A_j then contains the address of the register to be accessed. In either case, for indirect addressing, the contents of the address register becomes part of the command transmitted to shared resource 70.

A significant feature of the present embodiment is its ability to increment the contents of a shared B register "on the fly". This is important in eliminating steps required to increment a loop count in a task in which iterations of a loop are being performed by more than one processor. In previous machines, in order to perform a read and increment, a processor was required to issue

a "test and set" instruction to grab control of the necessary shared B register. This was followed by issuing a "read register" instruction to read the contents of the register and place it in a processor register. There the processor performed the increment and then issued a "write register" instruction to place the loop count back in the original shared B register. The processor clears the semaphore bit.

In the present embodiment, this array of instructions has been replaced with a single "read and increment" instruction. The "read and Increment" instruction causes read and increment circuit 94 to capture the loop count as it is read from shared registers 90, increment it and write the result back into the same shared B register. This operation is performed as a single sequence of events, eliminating contention from processors seeking the same variable and, therefore, removing the requirement to grab control of the register via a "test and set" semaphore command. The "read and increment" function leads to a savings in clock periods that would offer significant advantages in multiprocessing applications.

In the preferred embodiment, the bit-slicing of shared resource 70 into subcircuits 71 means that each read and increment circuit 94 must propagate its carry to its next most significant neighbor. In reality, due to the speed with which the calculation must be performed in order to save the result, it is necessary to generate a propagate line that is sent to all cards with bits more significant than the current card. Since the shared B registers are limited to 32 bits located on processor boards 0 through 7, this means that CPU0 must generate a propagate to CPU1 through CPU7 and CPU7 must be capable of accepting up to seven propagate lines and determining from them if it must perform an increment of its internal four bits. Since it is desireable for the processor boards to be identical, the basic processor board must be able to handle any combination of up to seven Carry_Ins and seven Carry_Outs.

In the preferred embodiment, command decoder 80 contains the circuitry necessary to individually control the I/O channels associated with the processor 202 on whose board it resides. Command decoder 80 generates I/O control signals and I/O demultiplexer 86 provides I/O addresses. Since each I/O address is 32 bits wide and only four bits can be transferred to a subcircuit 71 at a time, a multiplexing scheme is used in which the I/O address is transferred four bits at a time for eight consecutive clock periods. Operation of an I/O channel is illustrated for the sixteen processor case in FIG. 5. On the first three clock periods, the command nibbles are broadcast to all subcircuits 71. As illustrated, the second and third nibble transmitted contain the I/O channel number obtained from an address register A_j. The index j is determined from the j field in the instruction in CIP register 12. Following that broadcast, in the subsequent eight clock periods, the I/O address is broadcast four bits at a time to all subcircuits 71. The I/O address is retrieved from an address register A_k. Again, the index k is determined from the k field in the same instruction in CIP register 12. Each subcircuit 71 examines the I/O channel number received and determines if the channel number belongs to a channel on its processor board. If so, command decoder 80 on that processor board activates the channel and transfers the received I/O address to that channel.

In a like manner, an I/O address can be read from an I/O channel, formed into eight nibbles by multiplexer 96 and read back through read registers latch 88. This I/O interface functionality gives each subcircuit 71 the ability to control the I/O channels on its processor board.

In the preferred embodiment, a real time clock circuit 58 is provided within each local control circuit 10. Clock circuit 58 can be read by an instruction placed in CIP register 12 or loaded through read data latch 46 with the contents of a processor 202 scalar register S_j (where the index j is determined from the instruction in CIP register 12). Real time clock circuit 58 can only be loaded through shared resource circuit 70. Data from a scalar register S_j on one of the processors 202.1 through 202.N is written through write registers latch 78 and read registers selector 92 to read registers latch 88. From there it is broadcast to the clock circuit 58 on each of local circuits 10.1 through 10.N. The new starting time is loaded to each of the real time clock circuits 58 within the same clock period.

Although the present invention has been described with reference to the preferred embodiments, those skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

Claims

1. In a method of accessing data in an information register in a tightly coupled interprocessor communication system for a multiprocessor data processing system; wherein said communication system comprises a separate communications path (72, 74), a common shared resource circuit (70) connected to said path and distributed local control means (10) connected to each of a plurality of processors (202) and to the communications path (72, 74) for communicating and coordinating data transfer between said shared resource circuit and the connected processor; wherein said shared resource circuit includes common registers (90) including shared semaphore registers and shared information registers and wherein said local control means includes a local semaphore register (18) whose contents mirror the contents of an associated shared semaphore register, wherein the method is of the type wherein a local semaphore register bit associated with a desired shared information register is tested and, if the local semaphore register bit is not set, a bit is set in the associated shared semaphore reg-

ister corresponding to the local semaphore register bit, the desired shared information register is accessed through the local control means and the associated shared semaphore register bit is cleared, the improvement characterized by:

if, when the local semaphore bit is tested, the local semaphore register bit is found to be set, branching and executing instructions starting at a branch address.

2. In an interprocessor communication system in which access to a memory location (90) within a shared resource circuit (20) is shared by a plurality of processors (202), a method af automatically applying a function independent of the processors to modify the data in said location, wherein the function is applied as a result of a "read and modify" instruction received from one of said processors, the method comprising the steps of:

reading the data in said location;
capturing the data within the shared resource circuit as it is being sent to a requesting processor;
performing the function on the captured data to form a result;
storing the result back to said location.

3. An interprocessor communication system for a multiprocessor data processing system, comprising:

(a) a shared resource circuit including a plurality of clusters. each cluster including a shared semaphore register and a plurality of shared information registers;
(b) the shared resource circuit further including access control means for limiting access by each processor to the registers within a single cluster and autoincrement means for automatically incrementing data read from one of said registers and storing the result back into the same register;
(c) each processor including means for issuing instructions to access the common semaphore and information registers in said shared resource circuit;
(d) local control means connected to each processor and in relatively close proximity to its respective processor as compared to said common circuit, wherein said local control means includes a local semaphore register and issue control means for monitoring and controlling the issue of instructions requiring access to said shared resource circuit from the processor; and
(e) each of said local control means further including data control means for the transfer of data from its respective processor to a common register or from a common register to its respective processor.

4. The interprocessor communication system according to claim 3 wherein each local control means further includes command means for developing a command based on an issued instruction from its respective processor, said command being sent to said shared resource circuit in order to gain access to said shared circuit by the processor.

5. The interprocessor communication system according to claim 4 wherein each command means includes shared register address means for indirect addressing of the shared registers with the contents of a register connected to its respective processor.

6. The interprocessor communication system according to claim 5 wherein each local control means further includes a real time clock circuit accessible by its respective processor.

7. The interprocessor communication system according to claim 6 wherein each local control means further includes separate read and write paths connected to said shared resource circuit.

8. The interprocessor communication system according to claim 7 wherein each processor further includes address registers and scalar registers and each write path includes multiplexer means for selectively placing the contents of one of said command means, said address registers and said scalar registers on said data path.

9. The interprocessor communication system according to claim 8 wherein the shared resource circuit further includes I/O channel communication means for linking the shared resource circuit to an I/O channel and the local control circuit further includes means to transfer address information to said I/O channel communication means.

10. A method of forming an interprocessor communication system for transferring data and synchronizing activity between processors in a multiprocessor data processing system of N processors, comprising:

providing a common shared resource circuit including shared semaphore registers and shared information registers, wherein the shared information registers are usable for holding data to be accessed by a processor and the shared semaphore registers are usable for controlling access to a shared information register and for synchronizing activity between two processors;

partitioning the resource circuit into resource circuit blocks such that 1/N of the bits in each information register is placed in each block and a block is placed in relatively close proximity to each processor as compared to the other processors;
providing local control means connected to each processor, wherein said local control means is placed in relatively close proximity to its respective processor as compared to the remaining processors and said local control means includes issue control means and command means for coordinating communication between its respective processor and the shared resource circuit; and
providing interprocessor communication means for transferring commands and data between said local control means and said common shared resource circuit.

11. The method according to claim 10 wherein the method further includes providing command means connected to each local control circuit and each processor for forming an interprocessor communication command from an issued instruction.

12. The method according to claim 11 wherein the step of providing command means further includes providing means for forming a shared register address in said interprocessor communication command from the contents of a register attached to it's respective processor.

13. The method according to claim 12 wherein the method further comprises:

dividing said information registers into clusters, assigning one of said semaphore registers to each cluster; and
restricting access by each processor to those information and semaphore registers in their cluster.

14. The method according to claim 13 wherein the step of dividing the information registers into clusters further includes dividing the information registers into N+1 clusters, wherein each cluster contains the same number of information registers.

15. The method according to claim 14 wherein the step of dividing the information registers into clusters further includes restricting the number of information registers in each cluster to sixteen, wherein eight registers are used for scalar data and eight registers are used for address data.

16. An interprocessor communication system for a multiple processor computing system, comprising:

a shared information register;
a shared semaphore register including a bit used to control access to said shared information register;
a plurality of local circuits, wherein a local circuit is placed in close proximity and connected to an associated processor and wherein a local circuit includes:

a current instruction parcel register for receiving instruction parcels from the associated processor;
a real time clock;
a local semaphore register;
shared semaphore register monitoring means for monitoring changes in the shared semaphore register and reflecting those changes in the local semaphore register;
local semaphore testing means for testing a bit in said local semaphore register;
an instruction issue control connected to said local semaphore testing means and to each of the other local circuits for monitoring requests for interprocessor communication from other local circuits and for enabling the issue of instructions from the current instruction parcel register as a function of the state of a bit tested in its local semaphore register and of the requests received from other local circuits; and
control generation means connected to said current instruction parcel and said instruction issue control for converting issued instructions into a command, said control generation means including register address means for indirect addressing of the shared registers with the contents of a register connected to its respective processor; and

interprocessor communication means connected to said plurality of local circuits, said shared information register and said shared semaphore register for transferring a command from one of said local circuits to said shared registers in order to perform one of a group of functions including:

reading the shared information register;
writing the shared information register; and
loading the contents of the semaphore register into the local semaphore registers.

17. The interprocessor communication system according to claim 16 wherein the shared information register includes autoincrement means for automatically incrementing data read from said register and

storing the result back into the register and the group of functions performed by a command further includes reading and incrementing the contents of the shared register.

18. The interprocessor communication system according to claim 16 wherein the system further comprises I/O channel means connected to said interprocessor communication means for reading and writing to an I/O channel.

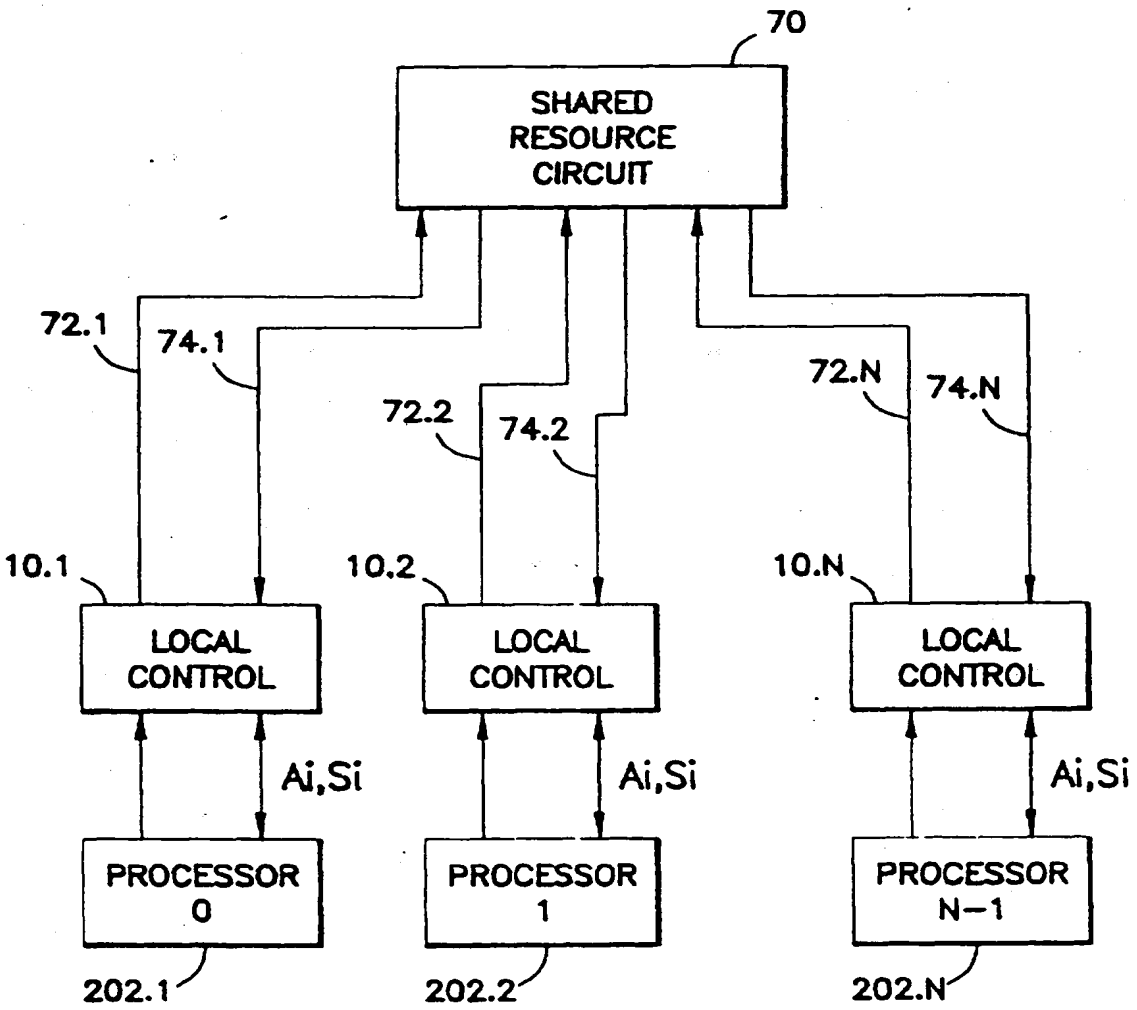

70
SHARED RESOURCE CIRCUIT
72.1
74.1
72.2
74.2
72.N
74.N
10.1
10.2
10.N
LOCAL CONTROL
LOCAL CONTROL
LOCAL CONTROL
Ai,Si
Ai,Si
Ai,Si
PROCESSOR 0
PROCESSOR 1
PROCESSOR N−1
202.1
202.2
202.N

FIG. 1

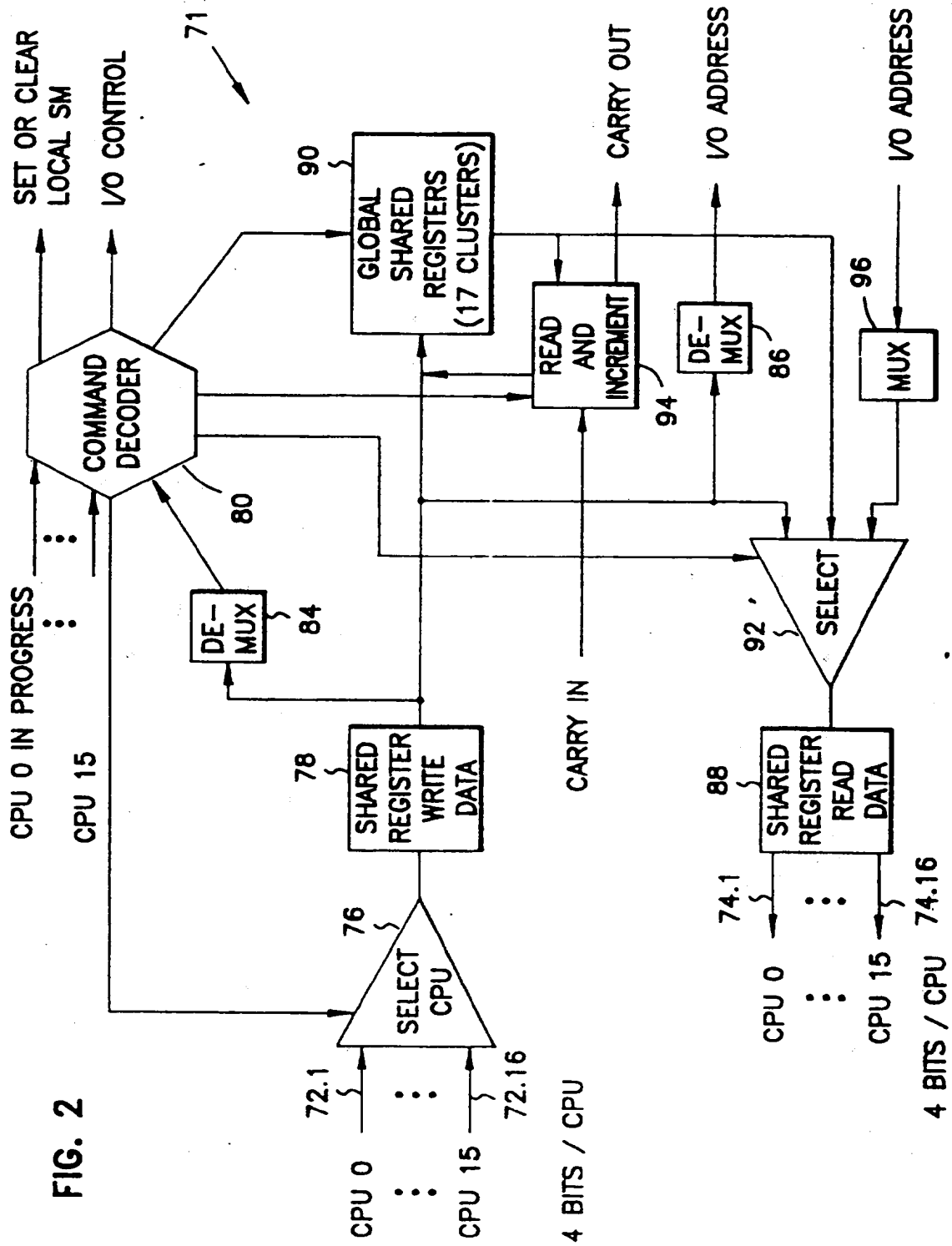
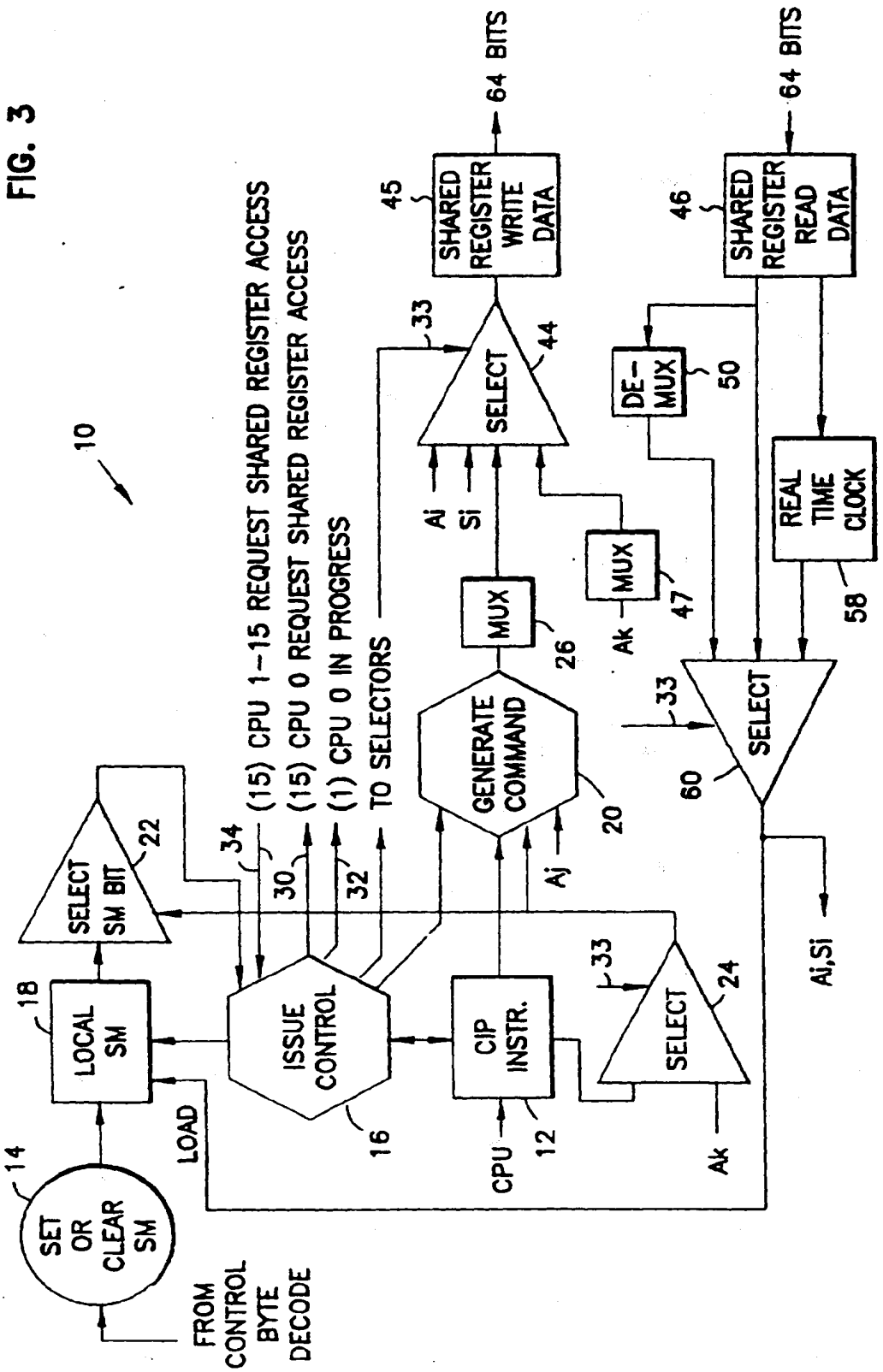

71
SET OR CLEAR LOCAL SM
I/O CONTROL
90
GLOBAL SHARED REGISTERS (17 CLUSTERS)
CARRY OUT
I/O ADDRESS
I/O ADDRESS
96
MUX
READ AND INCREMENT
94
DE-MUX
86
COMMAND DECODER
80
CPU 0 IN PROGRESS
CPU 15
DE-MUX
84
92'
SELECT
78
SHARED REGISTER WRITE DATA
CARRY IN
88
SHARED REGISTER READ DATA
76
SELECT CPU
74.1
74.16
72.1
72.16
CPU 0
CPU 15
4 BITS / CPU
CPU 0
CPU 15
4 BITS / CPU

FIG. 2

FIG. 3


10
FROM CONTROL BYTE DECODE
SET OR CLEAR SM
14
LOAD
LOCAL SM
18
SELECT SM BIT
22
ISSUE CONTROL
16
CPU
12
CIP INSTR.
GENERATE COMMAND
20
Aj
34
(15) CPU 1-15 REQUEST SHARED REGISTER ACCESS
30
(15) CPU 0 REQUEST SHARED REGISTER ACCESS
32
(1) CPU 0 IN PROGRESS
TO SELECTORS
MUX
26
Ai
Si
SELECT
44
33
45
SHARED REGISTER WRITE DATA
64 BITS
MUX
47
Ak
DE-MUX
50
SELECT
24
Ak
SELECT
60
33
46
SHARED REGISTER READ DATA
64 BITS
REAL TIME CLOCK
58
Ai,Si

RESOURCE SUBCIRCUIT PLACEMENT	CLOCK PERIOD		
	v	v+1	v+2
CPU 0	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^0 - 2^3$
CPU 1	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^4 - 2^7$
CPU 2	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^8 - 2^{11}$
CPU 3	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{12} - 2^{15}$
CPU 4	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{16} - 2^{19}$
CPU 5	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{20} - 2^{23}$
CPU 6	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{24} - 2^{27}$
CPU 7	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{28} - 2^{31}$
CPU 8	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{32} - 2^{35}$
CPU 9	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{36} - 2^{39}$
CPU 10	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{40} - 2^{43}$
CPU 11	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{44} - 2^{47}$
CPU 12	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{48} - 2^{51}$
CPU 13	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{52} - 2^{55}$
CPU 14	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{56} - 2^{59}$
CPU 15	$CMD^0 - CMD^3$	$CMD^4 - CMD^7$	$2^{60} - 2^{63}$

FIG. 4

RESOURCE SUBCIRCUIT PLACEMENT	CLOCK PERIOD						
	v	v+1	v+2	v+3	v+4		v+10
CPU 0	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$
CPU 1	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$
CPU 2	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$
CPU 3	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
CPU 13	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$
CPU 14	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$
CPU 15	$CMD^0 - CMD^3$	$Aj^0 - Aj^3$	$Aj^4 - Aj^7$	$Ak^0 - Ak^3$	$Ak^4 - Ak^7$	...	$Ak^{28} - Ak^{31}$

FIG. 5

EP 0 712 076 A2

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 712 076 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
06.11.1996 Bulletin 1996/45

(43) Date of publication A2:
15.05.1996 Bulletin 1996/20

(21) Application number: **96200086.5**

(22) Date of filing: **22.11.1991**

(51) Int Cl.[6]: **G06F 9/46**, G06F 15/16

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IT LI LU NL SE

(30) Priority: **14.02.1991 US 655296**

(62) Application number of earlier application in accordance with Art. 76 EPC: **92901473.6**

(71) Applicant: **CRAY RESEARCH, INC.**
Eagan, Minnesota 55121 (US)

(72) Inventor: **Schiffleger, Alan J.**
Chippewa Falls, Wisconsin 54729 (US)

(74) Representative:
Beresford, Keith Denis Lewis et al
BERESFORD & Co.
2-5 Warwick Court
High Holborn
London WC1R 5DJ (GB)

(54) System for distributed multiprocessor communication

(57) A tightly coupled communication scheme based on a common shared resource circuit and adapted particularly to a multiprocessing system including 2^N CPUs. A mechanism has been added that allows data in a shared register to be read and incremented as a single instruction, eliminating the need for semaphore manipulations during the operation. A second mechanism has been added to permit the use of indirect addressing in the addressing of semaphore bits and shared registers. Operating systems can relocate semaphore bits and message areas to permit simultaneous execution of the same function within a single task. In addition, an instruction has been added which tests of the semaphore bit and acts upon the state of that bit. If the semaphore bit is not set then the processor takes control of the semaphore bit by setting it. If the semaphore bit is set, the processor will execute a branch and execute other instructions. Thus, jobs assigned to a processor in a multiprocessing, multitasking application do not block or wait for the semaphore bit to clear.

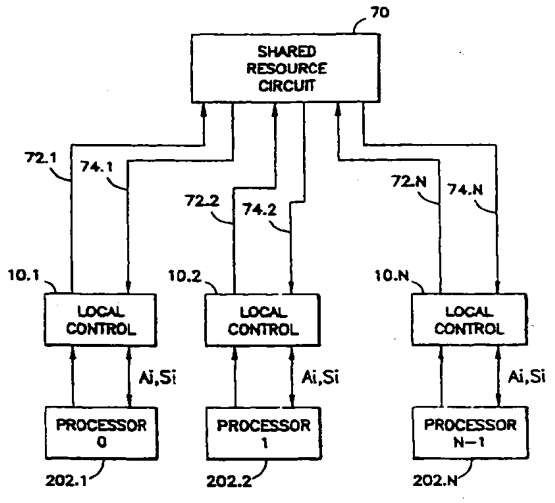


FIG. 1

EP 0 712 076 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 20 0086

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
D,X Y A	US-A-4 754 398 (PRIBNOW RICHARD D) 28 June 1988 * the whole document *	10-14,16 1 15,17,18	G06F9/46 G06F15/16
Y	US-A-4 725 946 (PRANGE PATRICK E ET AL) 16 February 1988 * column 3, line 18 - line 37 *	1	
A	EP-A-0 343 646 (HITACHI LTD ;HITACHI MICROCUMPUTER ENG (JP)) 29 November 1989 * claim 1 *	1	
A	EP-A-0 330 836 (CONVEX COMPUTER CORP) 6 September 1989 * page 1, line 1 - page 8, line 55 * * abstract; figures 1-4 *	10-18	
A	WO-A-83 04117 (WESTERN ELECTRIC CO) 24 November 1983 * page 1, line 1 - page 6, line 35 * * page 7, line 17 - page 17, line 8 * * page 31, line 3 - page 32, line 15 * * abstract; figures 1-4 *	10-18	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F
A	EP-A-0 351 556 (MODULAR COMPUTER SYST) 24 January 1990 * the whole document *	10-18	

[redacted]

Place of search	Date of completion of the search	Examiner
THE HAGUE	3 September 1996	Schenkels, P

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EP-96200086.5

CLAIMS INCURRING FEES

The present European patent application comprised at the time of filing more than ten claims.

☐ All claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for all claims.

☐ Only part of the claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims and for those claims for which claims fees have been paid.

namely claims:

☐ No claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims.

LACK OF UNITY OF INVENTION

The Search Division considers that the present European patent application does not comply with the requirement of unity of invention and relates to several inventions or groups of inventions.
namely:

See Sheet B.

☐ All further search fees have been paid within the fixed time limit. The present European search report has been drawn up for all claims.

☒ Only part of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the inventions in respect of which search fees have been paid.

namely claims: 1, 10 - 18

☐ None of the further search fees has been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the invention first mentioned in the claims.

namely claims:

European Patent Office

LACK OF UNITY OF INVENTION EP 96200086.5 - B -

The Search Division considers that the present European patent application does not comply with the requirement of unity of invention and relates to several inventions or groups of inventions, namely:

Claim: 1:

Semaphore bit is tested, if set then branch

Claims: 2-9:

Shared resource with read-modify instruction or autoincrement

Claims: 10-18:

Partitioning of the resource circuit and updating the local semaphore registers